Training Classifiers to Identify TCP Signatures in Scientific Workflows

George Papadimitriou*, Mariam Kiran‡, Cong Wang†, Anirban Mandal†, Ewa Deelman*

*Information Sciences Institute, University of Southern California, CA, USA
‡Lawrence Berkeley National Laboratory, CA, USA
†RENCI, University of North Carolina at Chapel Hill, NC, USA

***Abstract*—Identifying network anomalies is an important measure to ensure reliability and quality of data transfers among facilities. Scientific workflows in particular heavily rely on *good* network performance to ensure their smooth executions. In this paper, we present a lightweight classifier system that is able to recognize anomalous TCP transfers. Using random forest trees and labeled data sets, we evaluate the classifier with real workflow transfers for ground truth data. Our studies reveal that various TCP congestion algorithms behave differently in anomalous conditions. We show that training classifiers on these separately can aid detection in network performance deterioration. Results reveal that our classifiers are able to better predict anomalous flows for TCP Reno and Hamilton compared to Cubic and BBR, due to the manner how their congestion control algorithms handle the anomalies.**

***Index Terms*—TCP congestion algorithms, Cubic, Reno, Hamilton, BBR, Random forest tree classification**

I. Introduction

Today's large-scale science workflows often use Transmission Control Protocol (TCP) as a standard protocol to reliably transfer data over wide area networks. TCP is designed to cope with network congestion, to optimize throughput and packet delivery across large scale, inter-domain networking infrastructures [1] [2]. As demand for high-bandwidth increases [3], it is imperative to monitor and detect degradation in throughput and possible network bottlenecks causing congestion. One way to understand if there are network anomalies, is to study TCP behavior and monitor how it changes throughput and congestion window sizes to cope in an unfortunate situation. Passive analysis of TCP traces, such as more retransmission events, dropped throughput, longer congestion windows and longer return time to transmits (RTT), can all help provide indications of network performance problems and help resolve the issues [4].

Developed as a network-congestion avoidance algorithm, TCP offers congestion control in the network by signalling to the sender to slow-down or quicken data transfers depending on health of the link. TCP variants, such as Cubic, Reno, Hamilton, and more, have been designed to make TCP robust, while ensuring high performance [5]. In these techniques, TCP changes its internal settings (e.g. congestion window sizes, acknowledgement wait times, to name a few) to prioritize different metrics, such as high bandwidth (Cubic and Reno) or minimize loss (Vegas). A recent TCP variant, Google developed BBR helps achieve 4% higher network throughput for their YouTube services [6]. All TCP congestion algorithms are designed to retransmit packets if something wrong is detected during transfer.

Analyzing TCP behaviors in varying network conditions is of primary interest for network research. Network anomalies could cause performance degradation, such as being generated via sender being slow to transfer, client delaying acknowledgements, or just badly configured network links causing latency or packet loss [7]. TCP statistics have been explored to recognize network anomalies [4], predict throughput [8] or recognize congestion [9]. Current techniques have used statistical and machine learning approaches to ingest large amounts of TCP variables and to recognize common features. However, these techniques often only focus on just one TCP congestion algorithm and have limited study in how elephant and mice flows behave with TCP [10]. Additionally, end-host configurations are usually always hidden from WAN operators, which makes it difficult to study TCP behavior without knowing which one is being used.

In this paper, we make a number of novel contributions to TCP research. Firstly, we investigate in-depth how large and small file transfers are handled in normal and anomalous conditions, and secondly, we investigate many TCP-variants. Thirdly, we utilize random forest trees, a powerful machine learning technique that uses bagging, to partition large data sets into commonly identified features, for normal and anomalous conditions. These techniques allows us to address the following questions: (1) What are the unique behaviors of TCP variants in large and small file transfers, (2) How do these behaviors change in normal versus in presence of network anomalies - packet loss, duplication and reordering, and (3) can these behaviors be used to accurately identify anomalies in TCP transfers in other environments. We use random forest trees as they are a lightweight techniques to output common features and can easily be deployed. We extend our tests to TCP traces collected in a data center to test the validity of our trained classifiers.

We begin by gathering labeled data for our classifiers with in-house experiments collecting TCP traces of large (elephant) and small (mice) file transfers with 4 different

TCP variants - Cubic, Reno, Hamilton and BBR. These are the most commonly explored TCP variants for large science file transfers. We then test the validity of our classifiers by collecting additional TCP traces from a data-intensive science workflow, the 1000 Genome workflow [11], configured using the Globus service for WAN transfers. These traces also include artificially induced anomalies to collect behaviors in anomalous conditions. We evaluate our technique with unseen workflow transfers to measure the accuracy of our 4 classifiers - Cubic, Reno, Hamilton and BBR. Additionally, we show our techniques are able to adapt to unseen network environments and present results in understanding transfer behaviors in DOE data centers.

The rest of this paper is organized as follows: Section II introduces background information, such as TCP variants and elephant/mice TCP flows. In Section III and Section IV, we introduce the experimental setup for data collection, and the initial data analysis. We present the classifiers in Section V, and their evaluations in Section VI. Sections VII and VIII provides the discussions and related work. Finally, Section IX concludes this work.

II. Background

A. TCP Variants

TCP provides reliable and error-checked delivery of a data stream between sender and receiver. Most variants are a research effort into TCP extensions that can allow improvement of various network anomalies and enable congestion control.

Figure 1 describes how the TCP handshake works [4]. The sniffer or Tstat (TCP statistics) collector is located on both the Sender and Receiver, recording number of bytes moved, messages (ACK, FIN) or the time when segments cross. The sniffer is able to track variables in both directions, estimating RTT observed at flow start and completion. In addition to window sizes, number of retransmissions, the sniffer is able to record (more than) 133 variables per flow during the transfer. Researchers have used these statistics to classify normal versus anomalous segments and build classifier systems to recognize packet loss or alterations [4], [12].

Since the seminal work of Jacobson et al. [13] that established implementations of the modern TCP, a number of variants have been invented, some prioritizing throughput over loss prevention. In this paper, we focus on 4 of these variants:

1) TCP Cubic: This is used as the current default TCP algorithm in Linux [14]. Compared to other TCP algorithms, Cubic modifies the window size using a cubic function such to improve the throughput scalability over long distance flows. During normal conditions, Cubic aggressively increases the window size to allow throughput utilization to rise quickly, and then slowly as it reaches saturation point. Window growth is independent of the RTT recorded, allowing Cubic to achieve equitable allocations between multiple flows on a link.

2) TCP Reno: Incorporating fast retransmit and fast recovery, TCP Reno is known as the dominant congestion control algorithm in Internet applications [15]. Reno works by cutting the window size by half when loss is detected. if multiple

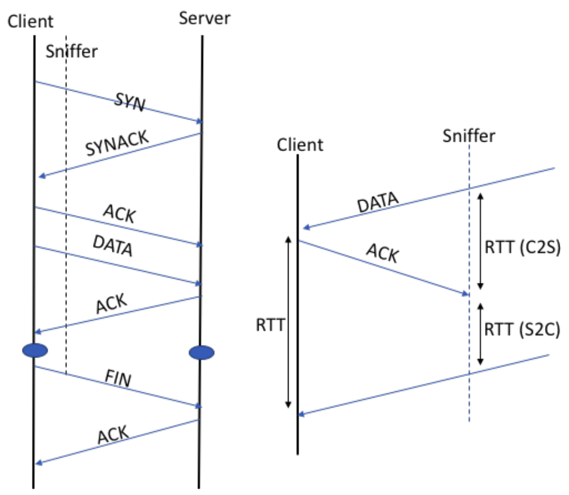


Fig. 1: Example of TCP Handshake. Adapted from [4].

ACKs are received, Reno performs fast retransmits, skipping the slow start phase by halving the congestion window. Similar to Cubic, Reno also increases window size if no loss occurs, but, different from Cubic has a shorter slow start phase allowing it to responds very quickly.

3) TCP Hamilton: Also known as H-TCP, similar to Cubic, this algorithm aggressively increases TCP throughput on high-bandwidth links, while maintaining RTT for smaller flows. Hamilton is seen to aggressively reduce congestion window whenever loss occurs, allowing it to recover quickly. New flows are seen to converge faster to optimal throughput under Hamilton than other algorithms [16], [17].

4) TCP BBR: This attempts to create a balance between fast re-transmits and fairness among flows on the same link. Rather than reacting on packet loss, as the previous versions do, BBR reacts on actual congestion and network models of the available bandwidth. Google introduced BBR to improve download speeds on internal networks [6]. It calculates RTT and bottleneck capacity and uses this to estimate its delivery rate. The values of RTT and bottleneck capacity are independently managed and BBR attempts to stay within this range.

Cubic is a far more efficient protocol for high-speed flows but not as aggressive as Reno and Hamilton. However these three versions are seen to utilize a much less available throughput as compared to BBR.

B. Network Anomalies that affect TCP Behavior

Network anomalies can create bogus traffic on the link causing network congestion and utilization in a router, impacting customer experience. Identifying these is imperative to improving network operations. Similar to designing TCP congestion algorithm, most efforts focus on early detection on packet loss and allowing the network to react. In TCP, congestion window size and RTT can give an indication of how throughput is performing, and as seen earlier, different TCP variants handle these differently. For example, Cubic is less aggressive on the congestion window size, calculating the size as a cubic function of time from the moment loss is observed. This results in two portions, the first being a concave, where the window rapidly grows until before the loss event occurred, and the convex region keeps probing for more bandwidth slowly at first and then rapidly. Cubic eventually reaches a stability between the concave and convex regions before looking for more bandwidth and growing its throughput [13]. This allows TCP to exhibit the wave behavior between the slow start, ramping up and coming down again.

Other than loss, there is significant effort to recognize congestion conditions such as overflowing buffers [2], [18]. However, there are commonly occurring network anomalies which can seriously impact user experience. Described by [4], [7], [19], there are three common network anomalies:

- **Packet Loss:** This occurs when one or more packets fail to reach their destination. These could either be caused by errors in transmission or too much congestion on link, causing routers to randomly drop packets.
- **Packet Duplication:** This occurs when the sender retransmits packets, thinking that the previous packets have not reached their destination. This is commonly seen when loss happen and retransmits increase.
- **Packet Reordering:** This is defined when arrival order of packets or sequence number is completely out-of-order. This is of particular relevance to real-time media streaming application, which show network instability.

C. Elephant and Mice (Long vs Short Transfers)

Internet and WAN network traces are seen to contain a mixture of flow characteristics. These can normally be divided into two groups - Elephant and Mice flows. For instance, a mice flow or small file transfers, are usually bursty and latency-sensitive, whereas big file transfers or longer (elephant) flows, are throughput-sensitive. If not managed efficiently, elephant flows can fill network buffers, causing queuing delays, packet drops and loss. On the other hand, mice flows are more difficult to predict and more dynamic in nature and can easily congest the network. Traffic engineering methods have explored separating these flows on different links to prevent congestion patterns and studying flow completion times [10].

D. Random Forest Trees

Random forest trees are a variant of decision tree classifiers that allow to recursively partition data sets into a rule space. The tree constructs a root that follows binary rules (true or false), to determine which rules are satisfied denoted by tree nodes and leaves. As we explore the tree, rules are extracted leading to terminal or decision nodes. Random forest allows to partition data samples on most distinguishing feature sets into purest samples possible. This measurement of purity is determined by the gini index or entropy function. In this study we use the gini index in our classification tree,

$$1 - \sum_{t=0}^{t=1} P_t^2 \tag{1}$$

where P denotes the probability of the sample belonging to a particular class. The summation in the equation represents two classes, but can be extended to multiple sample classes. The gini index is our cost function used to evaluate the class splits. During classification, the lower the gini index, means most samples belong to one class label.

Random forest classification [20] is a labeled machine learning technique that requires labeled data sets. Once trained, given a test data set, the classification can help predict labels for each test sample as it was trained.

III. Experimental Environment

To generate our labeled data set we used the ExoGENI testbed [21], which is a federated cloud testbed designed for experimentation and computational tasks. ExoGENI is orchestrated over a set of independent cloud sites located across US and connected via national research circuit providers through their programmable exchange points. By using a controlled experiment environment, we ensure that any system interference is minimized, and importantly, synthetically generated interference affects specific performance metrics targeted for evaluation. For our experiments, we created two types of environments (slices) on ExoGENI. The first was used to capture mice and elephant TCP flows, and the second one was used to capture real-like traces of a scientific workflow execution.

Figure 2 gives an overview of the first environment, which has one source-node, one forwarding-node and one destination-node. Each node has 4 2.2 Ghz vCPUs, 10 GB RAM and 75 GB storage, but all of them reside in a different ExoGENI region. The nodes are connected via a high speed layer 2 VLAN, and the round-trip time between source and destination is at ~21ms, while the forwarding-node is forwarding packets between the source and destination nodes.

Figure 3 presents an overview of the second environment, for real workflow executions on ExoGENI. This setup consists of one data-node, one forwarding-node, one master-node and four compute (worker) nodes (Figure 3). Each node has 4 2.2 Ghz vCPUs, 10 GB of RAM and 75 GB storage. Both compute and master nodes are co-located on the same site, while the data-node is spawned on a site in another region. The master and compute nodes communicate via the intra-site switch, while the data-node is connected over a high speed layer 2 VLAN, and the rountrip-time between master-node and data-node is at ~26ms. To facilitate the workflow execution, we configure our slice with HTCondor [22] and Pegasus WMS [11] on the master-node. Globus endpoints [23], [24] are created on both master-node and data-node to accommodate WAN transfers. Finally, the forwarding-node, forwards packets between the master and data nodes.

In both cases we use the Linux Traffic Control (TC) toolset [25] to introduce synthetic network and I/O anomalies, such as delay, packet loss and jitter, by configuring the Linux kernel packet scheduler. During the mice and elephant flow experiments we introduce these anomalies on both of the forwarding-node's interfaces. However, in the case of the real

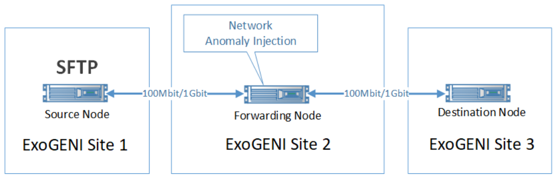


Fig. 2: Experimental setup on the ExoGENI testbed. SFTP tests for mice and elephant flows.

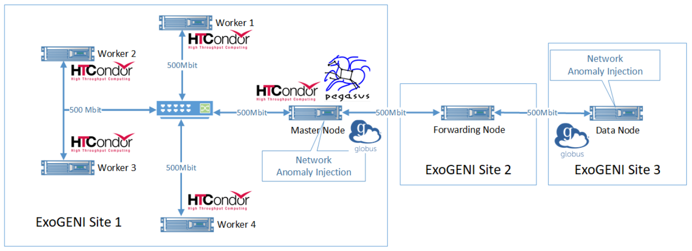


Fig. 3: Experimental setup on the ExoGENI testbed. Workflow tests using Globus.

scientific data transfers, we apply the anomalies on both the master and the data node, instead of the forwarding-node. To capture low level TCP statistics, we used the Tstat tool[1] to capture network traces on the source node in Figure 2, and on the data node of Figure 3.

A. Elephant and Mice Transfers

To generate labeled TCP data under normal and anomalous conditions for mice and elephant transfers we used the setup of Figure 2.

Mice flow. For the mice flows we aimed for 1000 SFTP transfers with a transfer size between 80MB and 120 MB, the link bandwidth is set to 1 Gbps among all the nodes.

Elephant flow. For the elephant flows we aimed for 300 SFTP transfers with a transfer size between 1 and 1.2 GB, the link bandwidth is set to 100 Mbps among all the nodes.

Table II summarizes the total number of flow samples collected for all TCP congestion algorithms for mice and elephant flows.

B. 1000 Genome Workflow Transfers

In addition to elephant and mice flow experiments, we collected traffic traces from a real science workflow named 1000-Genome project workflow that reconstructs genomes of 2,504 individuals across 26 different populations [26]. This workflow identifies mutational overlaps in data from the 1000 genomes project, providing a null distribution for rigorous statistical evaluation of potential disease-related mutations [27]. In our experiment, we recorded TCP statistics during this workflow setup phase shown in Figure 3. Table I summarizes the number of samples collected in our data-set.

C. Building the Training Data-set

Tstat collects 133 variables of TCP behavior in both directions: client-to-server and server-to-client. These features are listed on Tstat's documentation with their descriptions [28]. Using the bytes transferred and completion time traces, we can calculate the throughput in both directions.

[1]http://tstat.polito.it/

Type	TCP Congestion Algorithm			
	Cubic	Reno	Hamilton	BBR
Normal	257	265	258	221
Loss 1%	273	257	277	225
Loss 3%	281	277	289	0
Loss 5%	285	277	273	0
Dupl. 1%	265	265	265	225
Dupl. 5%	265	265	265	217
Reord. 25%	265	269	264	217
Reord. 50%	269	253	302	217

TABLE I: Number of flows generated at Data Node under normal and anomalous conditions.

Type	TCP Congestion Algorithm							
	Cubic		Reno		Hamilton		BBR	
	M	E	M	E	M	E	M	E
Normal	1000	300	1000	300	1000	300	1000	300
Loss 0.1%	1000	300	1000	300	1000	300	1000	300
Loss 0.5%	1000	300	1000	300	1000	300	1000	300
Loss 1%	1000	300	999	300	998	300	1000	300
Dupl. 1%	1000	300	1000	300	1000	300	1000	300
Dupl. 5%	1000	300	1000	300	1000	300	1000	300
Reord. 25%	1000	300	1000	300	1000	300	1000	300
Reord. 50%	1000	300	1000	298	1000	297	1000	299

TABLE II: Number of Mice and Elephant flows generated to train the classifiers under normal and anomalous conditions. M: Mice, E: Elephant.

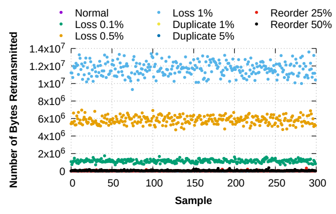
IV. Initial Analysis

A. Retransmissions

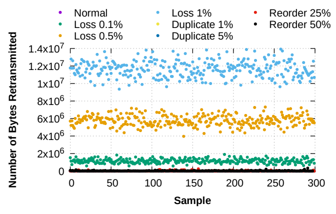
Figure 4 and 5 show the amount of retransmitted bytes, while using different TCP congestion algorithms and under non-anomalous and anomalous conditions in elephant and mice flows, respectively. The retransmitted bytes are seen to increase in packet loss experiments. In particular, they can be easily clustered among 0.1% (green), 0.5% (orange) and 1% (blue) packet loss rates in the case of elephant flows, while the cluster boundaries are vague in the case of mice flows. This is because elephant flow transfers are long enough to reflect the probability of packet loss events.

B. Throughput

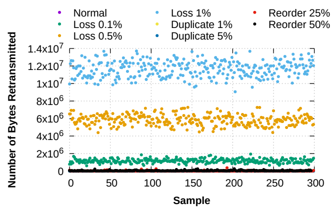
Figures 6 and 7 depict the throughput of the transfers while using different TCP congestion algorithms and under non-anomalous and anomalous conditions in elephant and mice flows. In elephant flows, Cubic (Fig. 6a), Reno (Fig. 6b) and Hamilton (Fig. 6c), present a decreasing throughput as packet loss is introduced, while BBR's throughput is not affected (Fig. 6d). In mice flows, the graphs show loss and reordering affect throughput greatly in all TCP variants. This implies that TCP congestion algorithms perform better for elephant flows rather than mice flows. This behavior could be attributed to TCP's slow-start phase, not giving enough time to mice flows to reach a steady state.



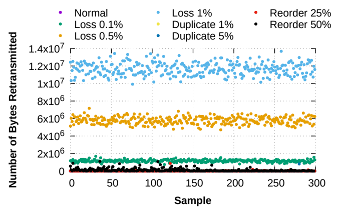
(a) Cubic



(b) Reno

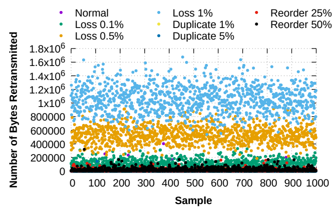


(c) Hamilton

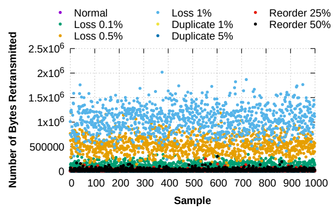


(d) BBR

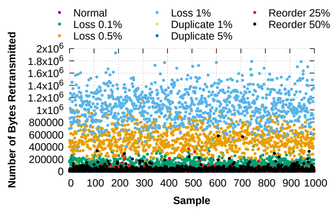
Fig. 4: Retransmissions in Elephant Flows.



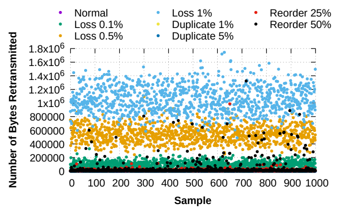
(a) Cubic



(b) Reno

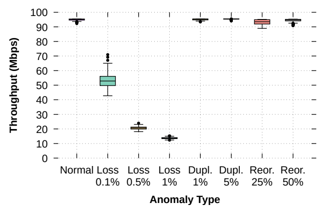


(c) Hamilton

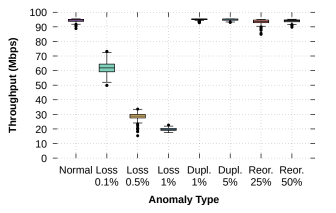


(d) BBR

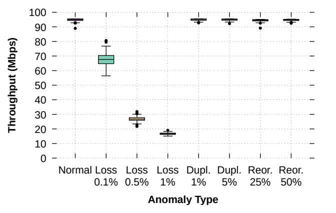
Fig. 5: Retransmissions in Mice Flows.



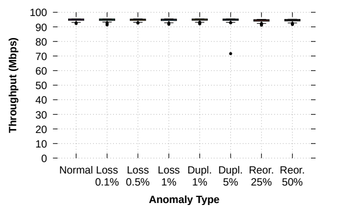
(a) Cubic



(b) Reno

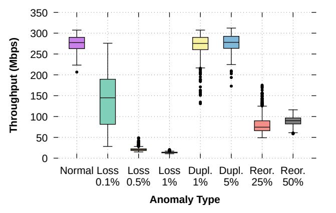


(c) Hamilton

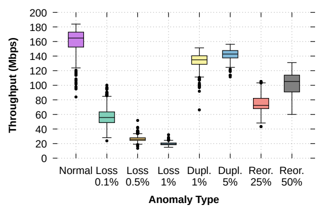


(d) BBR

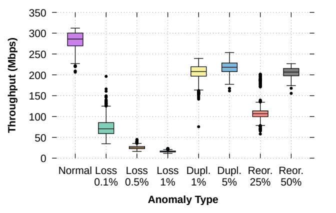
Fig. 6: Throughput of Elephant Flows. (Link speed: 100Mbps)



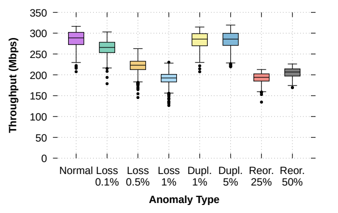
(a) Cubic



(b) Reno



(c) Hamilton



(d) BBR

Fig. 7: Throughput of Mice Flows. (Link speed: 1Gbps)

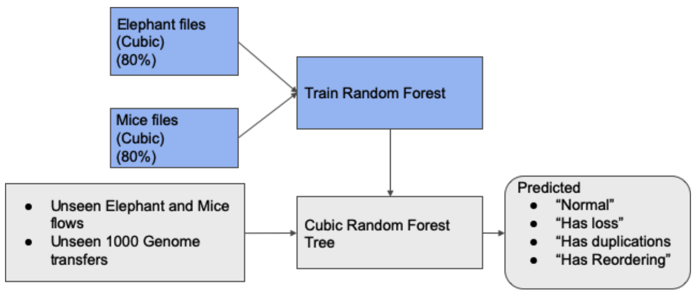


Fig. 8: Building Classifier for each TCP congestion variant.

V. Building Classifier

Figure 8 presents how the random forest classifiers were build. Because Cubic, Reno and Hamilton show similarities in how they behave under loss conditions, it is difficult to distinguish between them. As a result we build 4 individual classifiers (one for each TCP variant being investigated) and test each TCP variant separately.

A. Pruning the Trees

Constructing a tree with all possible decision rules is an NP-complete problem [29] and results in an overfitted tree. We combat this by using a greedy approach and limiting the tree to an optimum depth, number of leaf nodes and number of samples per leaf. This process is known as pruning the tree and is done to allow only important rules recognized in earlier tree branches, rather than letting it grow to less important leaf nodes[2]. We tuned the trees for each classifier by using 80%

[2]We used Scikit-learn's RandomForestTreeClassifier() to generate the trees.

of elephant and mice flows to achieve optimal classification accuracy.

B. Data Leakage

Data leakage is a common issue with building classifiers, when a particular feature shares same values between the training and test data. This feature causes 100% of training data to be partitioned into labeled classes, preventing the classifier to be generalizable for test data. In our initial training, we found that TCP features were recognizing specific value ranges distinguishing elephant and mice flows. This caused over-fitting to the test data, giving bad accuracy results. To combat this, we added randomly generated data to form 10% of the training data-set. This essentially turned the *pure* elephant and mice flow training data into impure data samples, but reduced the over-fitting problem of the classifiers, and improved their generalizability factor to test other flows.

VI. Evaluation

Figures 9 and 10 present the classification results on unseen elephant and mice flows, while Figure 11 shows the the results of the test workflow Tstat files. Each figure contains a subfigure for every classifier (Cubic, Reno, Hamilton, BBR), which presents the percentage of test data that were predicted as Normal, Loss, Duplication or Reordering (y-axis), given the anomaly type of test data (x-axis).

A. Testing data: Unseen Elephant and Mice flows

Elephant flow identification. Our initial analysis has revealed clear behavior distinctions when anomalies are found for elephant flows. While the BBR classifier is able to recognize all cases exactly (Fig. 9d), we find that Cubic and Reno classifiers are unable to recognize packet duplication cases (Fig. 9a and 9b). On the other hand, the Hamilton classifier recognizes only some (Fig. 9c). While loss is well recognized, due to the number of retransmits, packet duplication is not recognized in Tstat data. For reordering, we find that these flows have higher retransmission rates from the server side, being able to recognize 99% of reordered flows.

Mice flow identification. Mice flows are able to recognize duplication better than elephant flows. While Reno and Hamilton perform better here (Fig. 10b and 10c), Cubic and BBR are able to find 50% of the flows in the test data (Fig. 10a and 10d). This is because mice flows are able to record much higher retransmission in duplications, and quicker response than in elephant flows.

B. Testing data: Unseen Workflows transfers

Reno and BBR classifiers are not able to recognize loss and duplication, mistaking most for normal transfers (Fig. 11b and 11d). On the other hand, Cubic is able to classify flows under correct classes but also mark most as normal (Fig. 11a). The classifier for Hamilton is the best performing, but does not recognize reordering cases (Fig. 11c).

C. Testing data: Accuracy

Accuracy. In Figure 12 we have computed the accuracy of each classifier, and based only on the mice and elephant flows, the BBR classifier has a slight edge over the rest. However, taking the workflow data into consideration, the Hamilton classifier becomes the best performing one.

D. Testing with DTN Transfers

The classifiers were also tested against Tstat data collected at an OSG data-node[3] while transferring files to NERSC via the Globus service. All the classfiers, except Reno, predicted only reordering in the flows. On the other hand, Reno labeled some (~10%) of the flows as normal. Upon further investigation, we found that the DTN was configured to use TCP Reno as its TCP congestion algorithm, and despite the fact that the Reno classifier didn't recognize the majority of the flows as normal traffic, this result warrants further research and will form the basis of our future research direction on whether classifiers can be used to identify TCP algorithms.

VII. Discussion

Overall our classifiers are able to recognize packet reordering and loss well, but have struggled with duplication. We also found that the size of the file matters, giving different accuracy results between elephant and mice flows. The workflows and OSG transfers to NERSC were done using Globus, which converted the transfers to act similar to mice flows (due to the imposed parallelism), which was picked up by the classifiers.

Five most important features. Studying each of the trees in detail reveal the important features recognizing anomalous characteristics in all 4 classifiers. These were in the order of importance:

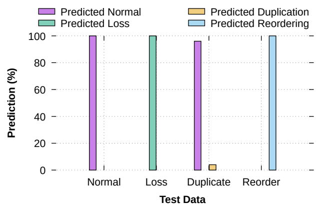
1) Actual throughput achieved during the transfer.
2) SACK option setting by the server and the client.
3) Standard deviation of RTT.
4) Number of segments with ACK field set to 1 by client.
5) First ACK segment (no SYN) recorded by server.

Unlike most current research only focusing on RTT and throughput, the classifiers are able find other features to help identify network problems.

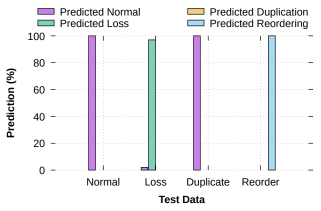
VIII. Related Work

Machine learning approaches are very useful to observe recurring phenomena (normal) and extract non-stationary transition patterns (anomalies) [19], [30]. For example, Mellia et al. [4] used passive TCP measurements to classify TCP anomalies, calculating high-level heuristics such as impact of flow length, return time observed and minimum RTT. Here, 7 possible causes of anomalies were recognized. Sundaresan et al. [9] showed that monitoring RTT during the slow start period and the difference between maximum and minimum RTT, can robustly identify loss congestion with 90% accuracy. Dukkipati et al. [10] argued that monitoring just flow

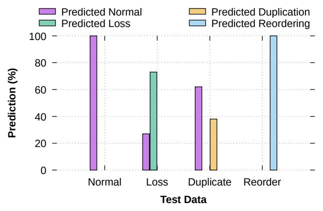
[3]Globus Endpoint of the OSG node used is osgconnect#stash



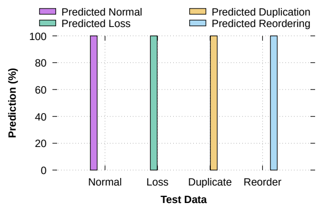
(a) Cubic



(b) Reno

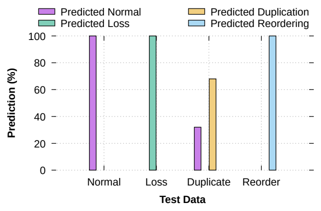


(c) Hamilton

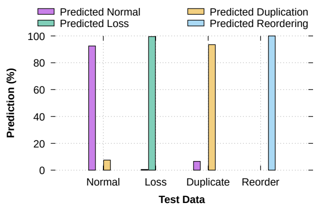


(d) BBR

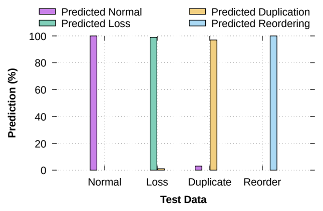
Fig. 9: Predictions for Elephant Flows.



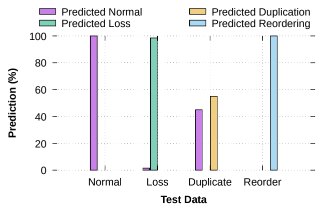
(a) Cubic



(b) Reno

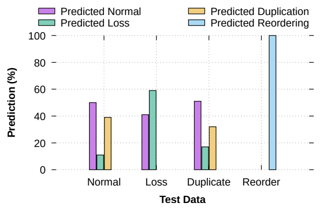


(c) Hamilton

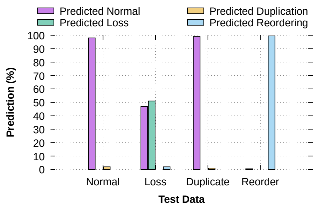


(d) BBR

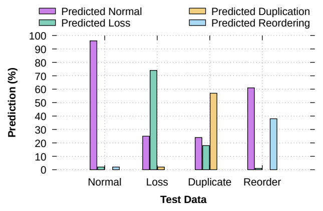
Fig. 10: Predictions for Mice Flows.



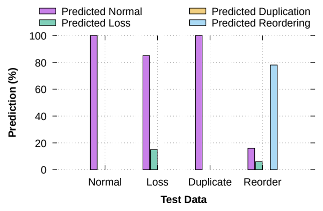
(a) Cubic



(b) Reno



(c) Hamilton



(d) BBR

Fig. 11: Predictions for the 1000 Genome Workflow Transfers.

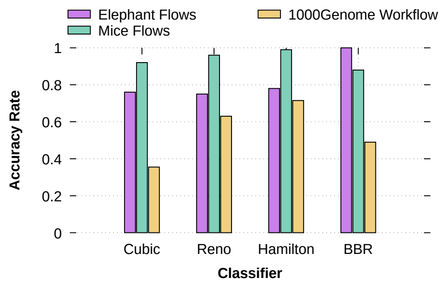


Fig. 12: Prediction Accuracy Rate.

completion time alone, can help improve congestion control in long lived TCP flows.

Network traces are often unlabeled data-sets and unsupervised feature extraction methods are more suitable [31]. However, to build correct classifier, one needs labeled data. Iglesias et al. [32] built a multistage feature selection method using filters and stepwise regression wrappers, to analyze 41 traffic features. Zanero et al. [33] used an unsupervised self organizing maps to identify patterns for intrusion detection on TCP payload. Others used TCP to predict throughput to detect bad behavior [8].

In their seminal work, Lakhina et al. [34] discussed how difficult diagnosing anomalies are, due to large amounts of high-dimensional and noisy data. Using principal component analysis, the authors showed how relational measurements can be used to isolate anomalous from normal traffic behaviors. The authors showed that on average anomalous traffic will have higher volumes when compared to their normal counterparts. Wang et al. [35] designed a multistage feature extraction method to filter information entropy from data through multiple stages to filter abnormal traffic. The authors improve false alarm rate hugely using this. Further, [36] used signal analysis of outages, flash crowds, attacks and measurement failures, based on IP and SNMP measurements. [37] used SVMs to predict anomalies in real time to work with DTNs and detect slow transfers, while [38] used genetic algorithms to select important features in TCP/IP packets.

Similar TCP signatures were detected by [39] by correlating flows and Markov models [40], while [9] developed TCP congestion signatures to identify self-induced and external congestion. [41] developed network signatures to correlate detection and unauthorized modifications. In recent works, [7] develop TCP classification for P4, but also found difficulty in identifying Reno and Cubic behaviors.

Effective modeling of TCP traffic can help differentiate normal and abnormal patterns. Carrascal et al. [42] used a combination of approaches, self organizing map and learning vector quantization to build intrusion detection systems. However, to the best of our knowledge, all current approaches focus on small number of features to build classifiers. In this work, we aim to use most of TCP variables.

IX. Conclusion

Studying TCP behavior can lead to develop new TCP congestion algorithms which may be game-changing for improving network transfer reliability [43]. Using random forest trees, our initial results show insights by identifying binary rules for key features. Being a white box approach, random forest reveals how the classifiers make decisions, showing features other than throughput, congestion window or completion time, can help identify anomalous transfers. Our experiments reveal that one needs to build separate classifiers for each TCP algorithm tracking and can help identify which algorithm is being used by the host machine. However, requiring more analysis, in future, we will extend the classifiers to identify TCP configurations and explore deep neural networks to help identify the feature relationships in TCP statistics.

Acknowledgment

This work was funded by DOE contract number #DE-SC0012636M, "Panorama 360: Performance Data Capture and Analysis for End-to-end Scientific Workflows". Any opinions, findings, or conclusions in this article are of the authors and do not necessarily reflect views of the sponsor.

References

[1] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science DMZ: A network design pattern for data-intensive science," *Conf. on High Performance Computing, Networking, Storage and Analysis*, 2013.

[2] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for tcp," *Commun. Surveys Tuts.*, vol. 12, pp. 304–342, July 2010.

[3] Cisco, "Trending analysis." https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html, 2019.

[4] M. Mellia, M. Meo, L. Muscariello, and D. Rossi, "Passive analysis of tcp anomalies," *Comput. Netw.*, vol. 52, pp. 2663–2676, Oct. 2008.

[5] T. V. Lakshman and U. Madhow, "The performance of tcp/ip for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. Netw.*, vol. 5, pp. 336–350, June 1997.

[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, pp. 50:20–50:53, Oct. 2016.

[7] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data plane performance diagnosis of tcp," in *Symposium on SDN Research*, (New York, NY, USA), pp. 61–74, ACM, 2017.

[8] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to tcp throughput prediction," *IEEE/ACM Trans. Netw.*, vol. 18, pp. 1026–1039, Aug. 2010.

[9] S. Sundaresan, A. Dhamdhere, M. Allman, and k. claffy, "Tcp congestion signatures," in *App. Net. Research Wksp.*, pp. 18–18, 2018.

[10] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, pp. 59–62, Jan. 2006.

[11] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[12] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi, "Traffic analysis with off-the-shelf hardware: Challenges and lessons learned," *Comm. Mag.*, vol. 55, pp. 163–169, Mar. 2017.

[13] V. Jacobson, "Congestion avoidance and control," in *Sym. Proc. Communications Architectures and Protocols*, pp. 314–329, 1988.

[14] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, July 2008.

[15] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling tcp reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 133–145, April 2000.

[16] T. Lukaseder, L. Bradatsch, B. Erb, R. W. Van Der Heijden, and F. Kargl, "A comparison of tcp congestion control algorithms in 10g networks," in *Conf. on Local Computer Networks*, pp. 706–714, Nov 2016.

[17] D. Miras, M. Bateman, and S. Bhatti, "Fairness of high-speed tcp stacks," in *Int. Conf. on Adv. Information Networking and Applications*, pp. 84–92, March 2008.

[18] A. Parichehreh, S. Alfredsson, and A. Brunstrom, "Measurement analysis of tcp congestion control algorithms in lte uplink," *Network Traffic Measurement and Analysis Conf.*, 2018.

[19] M. Ahmed, A. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[20] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[21] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, and J. Chase, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *Testbeds Research Infrastructure. Development of Networks Communities*, 2012.

[22] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience.," *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[23] www.globus.org, *globus*, 2018 (accessed February 24, 2019). https://www.globus.org.

[24] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in *ACM/IEEE Conference on Supercomputing*, 2005.

[25] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, vol. 213, 2002.

[26] 1000 Genomes Project Consortium, "A global reference for human genetic variation," *Nature*, vol. 526, no. 7571, pp. 68–74, 2012.

[27] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, and M. Atkinson, "Using simple pid controllers to prevent and mitigate faults in scientific workflows," in *Workflows in Support of Large-Scale Science*, pp. 15–24, 2016.

[28] Telecommunication Networks Group - Politecnico di Torino, "Tstat: Log tcp complete," 2016.

[29] R. Quinlan, *Learning efficient classification procedures, Machine Learning: an artificial intelligence approach*. Morgan Kaufmann, 1983.

[30] F. Palmieri and U. Fiore, "Network anomaly detection through nonlinear analysis," *Computers & Security*, vol. 29, no. 7, pp. 737–755, 2010.

[31] S. Zanero, "Analyzing tcp traffic patterns using self organizing maps," in *Image Analysis and Processing*, pp. 83–90, 2005.

[32] F. Iglesias and T. Zseby, "Analysis of network traffic features for anomaly detection," *Machine Learning*, vol. 101, pp. 59–84, Oct 2015.

[33] S. Zanero, "Analyzing tcp traffic patterns using self organizing maps," in *Int. Conf. on Image Analysis and Processing*, pp. 83–90, 2005.

[34] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *SIGCOMM*, pp. 219–230, 2004.

[35] H. Wang, Z. Gong, Q. Guan, and B. Wang, "Detection network anomalies based on packet and flow analysis," *Int. Conf. on Networking*, 2008.

[36] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *SIGCOMM Work. on Internet Measurement*, pp. 71–82, 2002.

[37] A. Syal, A. Lazar, J. Kim, A. Sim, and K. Wu, "Automatic detection of network traffic anomalies and changes," in *Systems and Network Telemetry and Analytics*, (New York, NY, USA), pp. 3–10, ACM, 2019.

[38] T. Shon, X. Kovah, and J. Moon, "Applying genetic algorithm for classifying anomalous tcp/ip packets," *Neurocomputing*, vol. 69, no. 16, pp. 2429 – 2433, 2006. Brain Inspired Cognitive Systems.

[39] G. Fernandes and P. Owezarski, "Automated classification of network traffic anomalies," in *Security and Privacy in Communication Networks*, pp. 91–100, 2009.

[40] G. Münz, H. Dai, L. Braun, and G. Carle, "Tcp traffic classification using markov models," in *Traffic Monitoring and Analysis*, pp. 127–140, 2010.

[41] K. Tan and B. Collie, "Detection and classification of tcp/ip network services," in *Annual Computer Security App. Conf.*, 1997.

[42] A. Carrascal, J. Couchet, E. Ferreira, and D. Manrique, "Anomaly detection using prior knowledge: application to tcp/ip traffic," in *Artificial Intelligence in Theory and Practice*, pp. 139–148, Springer US, 2006.

[43] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *SIGCOMM*, pp. 123–134, 2013.